Penentuan Jarak Terpendek dan Jarak Terpendek Alternatif Menggunakan Algoritma Dijkstra Serta Estimasi Waktu Tempuh

Asti Ratnasari[1], Farida Ardiani[2], Feny Nurvita A.[3]
Magister Teknik Informatika, Universitas Islam Indonesia, Yogyakarta 55584
Email : azz.informatic@gmail.com, [2] ardianifarida@gmail.com, [3] feny.nurvita@gmail.com

ABSTRAK

Algoritma dijkstra merupakan salah satu algoritma yang digunakan untuk mencari jarak terpendek dalam suatu graf. Prinsip greedy (serakah) oleh algoritma dijkstra digunakan untuk memecahkan masalah jalur terpendek pada sebuah graf. Implementasi sistem yang dibuat digunakan untuk mencari jarak terpendek, jarak terpendek alternatif serta estimasi waktu tempuh dalam sebuah graf. Hasil yang didapatkan dari implementasi sistem ini adalah mampu menemukan jarak terpendek dan jarak terpendek alternatif ketika terjadi hambatan (pemblokiran jalan) pada jalur terpendek utama dan juga dapat mengetahui estimasi waktu tempuhnya.

Kata Kunci : algoritma dijkstra, greedy, jarak terpendek, waktu tempuh

1. PENDAHULUAN

Setiap orang dalam melakukan perjalanan pasti memilih jarak terpendek untuk mencapai tujuannya, karena dapat menghemat waktu, tenaga serta bahan bakar. Kesulitan menentukan jarak terpendek timbul karena terdapat banyak jalur alternatif yang ada dari suatu daerah ke daerah lain dan juga memungkinkan memilih jalur alternatif apabila terdapat suatu hambatan pada jalur terpendek utama. Kebutuhan untuk menemukan jarak terpendek dan waktu tempuh tercepat tentunya juga diperhitungkan untuk menghindari kerugian seperti contoh bagi sebuah industri, kebutuhan untuk segera sampai tempat tujuan tepat waktu bahkan diharapkan bisa lebih cepat sangtlah dibutuhkan mengingat persaingan industri yang mementingkan kepuasan pelanggan dan menghindari kerugian karena kerusakan barang, hal itu dapat saja terjadi bila terjadi pemblokiran jalan secara tiba-tiba pada jalan yang seharusya dilalui, selain untuk industri jarak terpendek juga dibutuhkan untuk menghemat waktu tempuh bagi wisatawan yang ingin bepergian ke tempat wisata.

Pada kenyataannya kita dapat mengetahui jarak antar daerah menggunakan peta konvensional, akan tetapi peta konvensional tidak mampu menemukan jarak terpendek secara langsung, untuk itu dibangun suatu sistem yang dapat membantu menemukan jarak terpendek dan jarak terpendek alternatif serta estimasi waktu tempuh yang disajikan secara sederhana serta terkomputerisasi.

Pada penelitian ini digunakan algoritma dijkstra untuk penyelesaian penentuan jarak terpendek dan jarak terpendek alternatifnya. Algoritma dijkstra dipilih karena memiliki beberapa kelebihan selain menguntungkan dari segi *running time*, dijkstra dapat menyelesaikan beberapa kasus pencarian jalur terpendek, yaitu:

1. Pencarian jalur terpendek antara dua buah simpul tertentu (*a pair shortest path*).
2. Pencarian jalur terpendek antara semua pasangan simpul (*all pairs shortest path*).
3. Pencarian jalur terpendek dari simpul tertentu ke semua simpul yang lain (*single-source shortest path*).
4. Pencarian jalur terpendek antara dua buah simpul yang melalui beberapa simpul tertentu (*intermediate shortest path*) [1].

Cara lain untuk menemukan jarak terpendek dapat juga menggunakan algoritma ford dan algoritma floyd. Algoritma ford adalah salah satu algoritma yang dalam penyelesaian penentuan jarak terpendeknya untuk graf berarah yang salah satu simpulnya bisa memiliki bobot negatif, pencarian dalam algoritma Bellman-Ford melacak keseluruh simpul dan menentukan jalur yang terpendek [2], sedangkan algoritma floyd adalah algoritma Floyd-Warshall, adalah sebuah algoritma untuk mencari jalur terpendek memandang solusi yang diperoleh sebagai sebuah keputusan yang saling terkait. Algoritma ini hanya digunakan untuk graf berarah. Sama halnya seperti algoritma Bellman-Ford, algoritma ini juga memungkinkan memiliki bobot dengan nilai negatif dan mendeteksi simpul yang bernilai negatif jika ada. Setelah tidak ada simpul benilai negatif, eksekusi tunggal dilakukan untuk menemukan jalan terpendek antara semua pasangan simpul. Algoritma ini ditemukan secara independen oleh Bernard Roy pada tahun 1959, Robert Floyd pada tahun 1962, dan oleh Stephen Warshall pada tahun 1962 [3].

Penelitian ini menerapkan Pemrograman Berorientasi Objek (PBO) untuk pembuatan sistemnya, dengan Delphi sebagai bahasa pemrograman dan My SQL sebagai penyimpanan basis datanya. PBO adalah teknik pemrograman yang menyediakan

mekanisme untuk mendukung pemrograman berorientasi objek dengan baik untuk menyelesaikan sebuah masalah [4]. Borland Delphi merupakan pemrograman berorientasi objek di lingkungan pemrograman visual untuk mengembangkan aplikasi 32-bit untuk ditempatkan pada Windows dan Linux. Bahasa pemrograman ini dapat membuat aplikasi yang sangat efisien dengan minimal coding manual [5]. Keunggulan bahasa pemrograman ini terletak pada produktifitas, kualitas, pengembangan perangkat lunak, kecepatan kompilasi, pola desain yang menarik serta dengan pemrogramannya yang terstruktur, selain itu dapat digunakan untuk merancang program aplikasi yang memiliki tampilan seperti program aplikasi lain yang berbasis Windows. Borland delphi merupakan perangkat lunak yang mendukung konsep pemrograman berorientasi objek karena di dalamnya terdapat fasilitas pembuatan kelas dan objek, pembuatan konstruktor dan destruktor, mendukung konsep polimorfisme dan fasilitas-fasilitas pendukung PBO lainnya.

2. METODE PENELITAN

Pengembangan sistem informasi ini dilakukan dengan beberapa tahapan yaitu:
1. Studi literatur dilakukan untuk mempelajari teori-teori yang berkaitan dengan penelitian, yaitu algoritma dijkstra dan pencarian waktu tempuh.
2. Analisis kebutuhan meliputi kebutuhan hardware dan software untuk pengembangan sistem.
3. Perancangan dalam sistem ini meliputi perancangan proses data berupa desain (*Data Flow Diagram*) DFD dan (*Entity Relationship Diagram*) ERD, perancangan database digunakan untuk mengetahui relasi antar tabel-tabel yang berada dalam database, perancangan tabel bertujuan untuk mengetahui lebih rinci isi dari tiap tabel yang ada dalam database, perancangan *user interface* digunakan untuk mempermudah *user* dalam mengoperasikan system, perancangan kendali berupa peringatan-peringatan yang akan disertakan dalam sistem, yang berujuan untuk memberitahu *user* apabila terjadi kesalahan agar *user* mampu mengetahui letak kesalahan dan mempermudah untuk melakukan perbaikan.
4. Implementasi, penerapan pengembangan aplikasi ini berdasarkan desain/perancangan sistem yang dihasilkan.
5. Pengujian dilakukan untuk mengetahui kesesuaian hasil implementasi dengan hasil dari identifikasi kebutuhan, serta harapan dan tujuan pengembangan perangkat lunak penentuan jarak terpendek, jarak terpendek alternatif dan estimasi waktu tempuh.

3. PEMBAHASAN

Algoritma Dijkstra ditemukan oleh Edsger Dijkstra pada tahun 1959, adalah algoritma pencarian graf yang memecahkan masalah jalur terpendek yang bersumber dari satu simpul untuk sebuah graf dengan bobot simpul tidak boleh negatif [6]. Analisis dilakukan dengan cara memeriksa simpul dengan bobot terkecil dan memasukkannya ke dalam himpunan solusi dengan awal pencarian simpul asal membutuhkan pengetahuan tentang semua jalur dan bobotnya, sehingga dibutuhkan pertukaran informasi dengan semua simpul. Algoritma dijkstra memiliki sifat yang sederhana dan lempeng (straightforward), sesuai dengan prinsip kerja greedy. Algoritma *greedy* adalah algoritma yang memecahkan masalah langkah demi langkah, pada setiap langkah:
1. Mengambil pilihan yang terbaik yang dapat diperoleh saat itu
2. Berharap bahwa dengan memilih optimum lokal pada setiap langkah akan mencapai optimum global. Algoritma *greedy* mengasumsikan bahwa optimum lokal merupakan bagian dari optimum global [7].

Algoritma dijkstra dapat menyelesaikan jalur terpendek dari sebuah titik asal dan titik tujuan dalam suatu graf berbobot $G = (V, E)$. Jarak terpendek diperoleh dari dua buah titik jika total bobot dari semua simpul dalam jaringan graf adalah yang paling minimal.
Sebelum meneruskan lebih lanjut perlu diketahui beberapa notasi yang digunakan:
1. l(i,j): panjang jarak antara simpul i ke simpul j
2. a : titik awal pencarian
3. d_{ai} : jarak yang dikenal terpendek dan bersifat permanen dari titik awal ke titik (i) dalam suatu jaringan.
4. q_i : titik terdahulu yang dikenal terpendek dari titik awal ke titik (i) dalam suatu jaringan.
5. c : titik terakhir yang telah pindah ke keadaan permanen.

Langkah prosedural algoritma dijkstra sebagai berikut :
1. Proses dimulai dari titik awal (a), $d_{aa} = 0$. Maka titik yang satu-satunya permanen adalah titik a. Dan titik lain berlabel sementara yang diisi dengan inisial tak hingga (∞).
2. Memeriksa cabang titik yang keluar dari titik terakhir yang bersifat permanen dengan persamaan dengan persamaan :

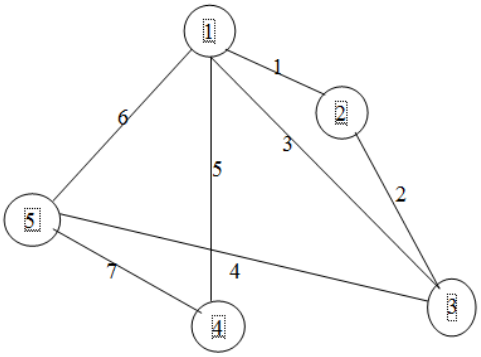
$$d_{ai}=\min[d_{ai},d_{ac}-l(c,i)] \quad (1)$$

3. Menentukan titik mana yang akan lulus dari label sementara menjadi label permanen. Dengan cara membandingkan nilai titik dari hasil langkah kedua dan diambil nilai titik paling minimum. Kemudian untuk mengetahui nilai titik permanen terdahulu, masukkan kedalam persamaan:

$$[d_{ai}\text{-}l(i,j)]=...d_{ai} \quad (2)$$

4. Setelah mendapatkan titik dengan nilai paling minimum, maka titik yang bernilai minimum tersebut ditetapkan sebagai titik permanen berikutnya.
5. Jika masih terdapat titik yang belum berlabel permanen maka ulangi kembali dari langkah 2.

Di bawah ini diberikan contoh sebuah graf tak berarah yang terdiri dari 5 buah titik dan 7 buah jalur yang menghubungkan antar dua buah titik. Algoritma djikstra digunakan untuk mencari jarak terpendek dari sebuah titik ke titik lainnya pada graf tak berarah tersebut.



Gambar 1: Contoh graf tak berarah

Berdasarkan contoh graf tak berarah di atas ditentukan titik awal pencarian adalah titik 1 dengan tujuan yaitu titik 4 dan akan dicari jarak terpendek yang dapat ditempuh dari titik 1 untuk menuju titik 4. Berikut ini tabel penjelasan graf menggunakan algoritma dijkstra:

Tabel 1: Penjelasan graf menggunakan algoritma dijkstra

Jalur	**Initial Jalur**					**Titik**	**l(i,j)**				
	1	**2**	**3**	**4**	**5**						
	0	0	0	0	0		**1**	**2**	**3**	**4**	**5**
1	1	0	0	0	0	**1**	∞	∞	∞	∞	∞
1-2	1	1	0	0	0	**2**	1	∞	∞	∞	∞
2-3	0	1	1	0	0	**3**	3	2	∞	∞	∞
3-5	0	0	1	0	1	**4**	5	∞	∞	∞	7
5-4	0	0	0	1	1	**5**	6	∞	4	∞	∞

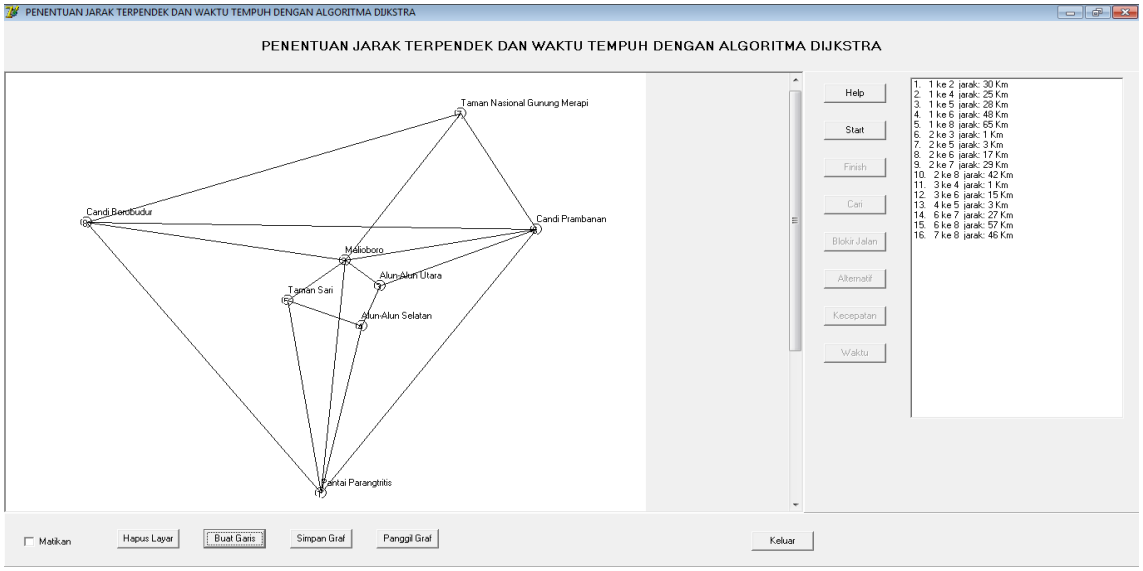
Penyelesaian algoritma dijkstra jalur titik 1 ke titik 4 telah diselesaikan seperti pada penjelasan dan tabel di atas menurut perhitungan penelusuran graf sesuai langkah prosedural algoritma dijkstra. Pada baris pertama semua *successor* di set 0 artinya untuk memberi nilai pada sumber titik rute yang akan dijadikan rute dan ketidakterbatasaan untuk semua titik lain, yang menyatakan fakta bahwa tidak diketahui lintasan manapun.

Untuk selanjutnya karena titik 1 sebagai sumber lintasan maka sudah pasti terpilih. Sehingga status set 0 berubah menjadi 1. Titik 1 akan cek titik yang bertetanggaan langsung yaitu titik 2, 3, 4 dan 5. Dari situ dijkstra akan memilih yang mempunyai bobot terendah untuk menuju titik selanjutnya. Terpilih titik 2 dengan bobot 1, set status 0 berubah menjadi 1 dan seterusnya. Maka dari pencarian jarak terpendek di atas, didapat lintasan yang terpendek berdasarkan pencarian dijkstra dari titik 1 ke 4 adalah melalui titik 1 langsung titik 4 dengan bobot lintasan 5.

4. UJI COBA SISTEM

Uji coba sistem dilakukan dengan membuat sebuah graf tak berarah dengan studi kasus *non-real* beberapa tempat wisata di Yogyakarta. Setiap titik dibuat oleh *user* dengan cara klik kanan *mouse* pada tempat yang tersedia dan mengisikan nama titik sesuai dengan kebutuhan tempat yang akan dibuat, kemudian *user* dapat membuat garis dengan klik tombol “Buat Garis” dan

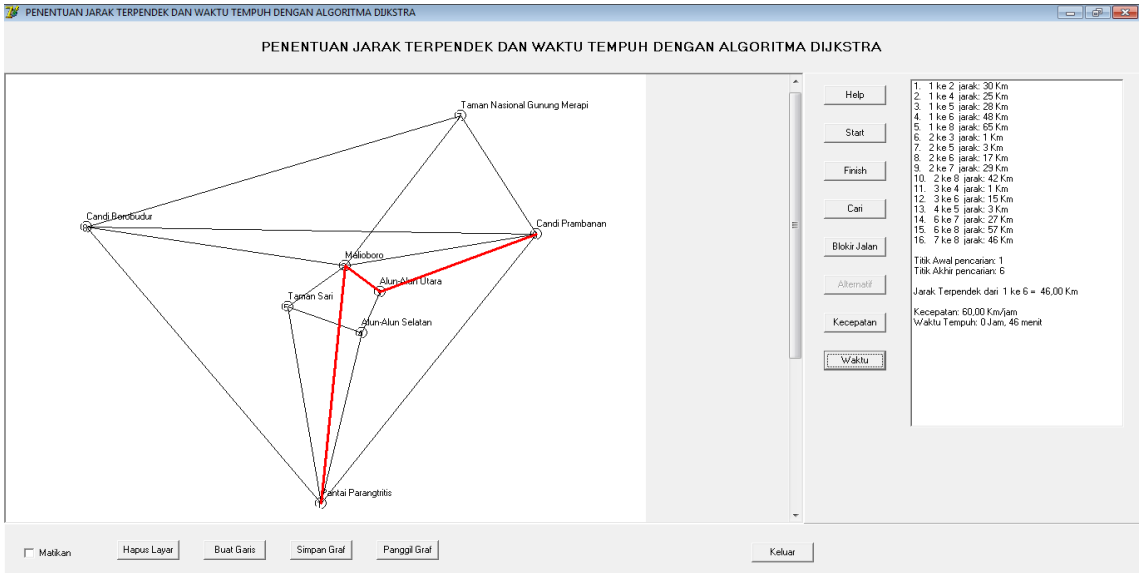
mengisikan titik pangkal, titik akhir, dan bobot yang kemudian akan muncul pada *listbox*. Bobot merupakan jarak antara titik pangkal dengan titik akhir dengan satuan Km, dilakukan hal yang sama untuk proses pembuatan titik dan jarak berikutnya sampai graf sesuai dengan kebutuhan *user*. Contoh tampilan graf tak berarah dengan studi kasus tempat wisata kota Yogyakarta dapat dilihat pada gambar 2 di bawah ini.



Gambar 2 : Tampilan sebuah graf dan jarak ditampilkan dalam *listbox*

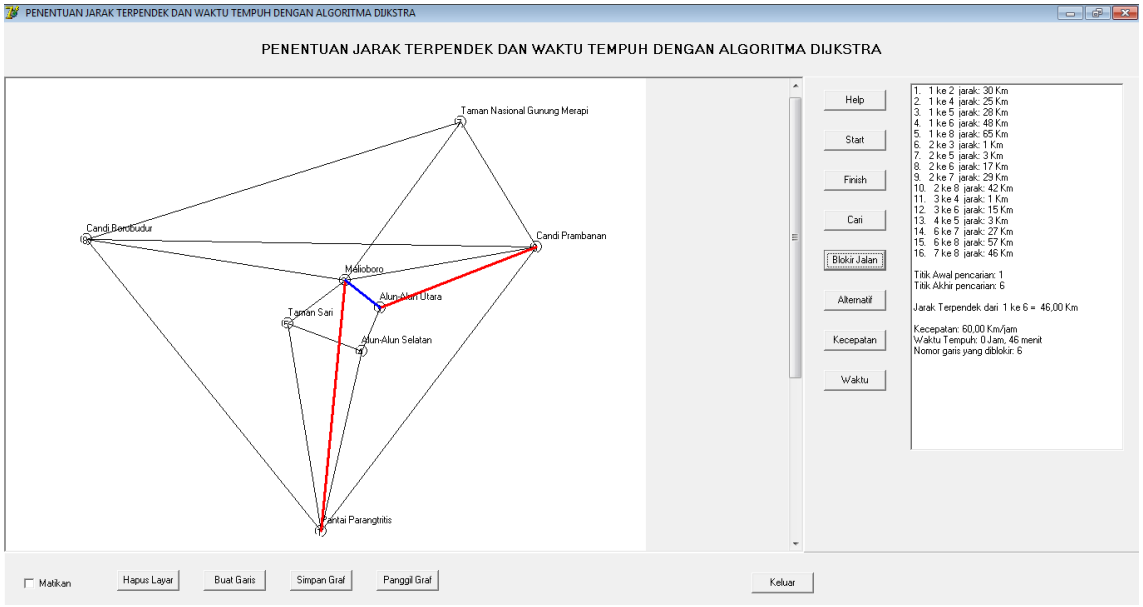
Setelah graf selesai dibuat seperti yang terlihat pada Gambar 2, jika ingin mengetahui jarak terpendek maka *user* diminta untuk mengisikan titik awal pencarian sebagai posisi awal dan titik akhir pencarian sebagai tujuan. *User* harus klik tombol "Start" untuk memasukkan titik awal pencarian dan klik tombol "Finish" untuk memasukkan titik akhir pencarian. Kemudian untuk mengetahui jarak terpendek, *user* harus klik tombol "Cari". Studi kasus ini dimisalkan *user* ingin ke Candi Prambanan dari Pantai Parangtritis, maka *user* hanya perlu memasukkan titik awal pencarian, yaitu Pantai Parangtritis dan titik akhir pencarian yaitu Candi Prambanan, maka akan diperoleh jarak terpendek dari seluruh jalur yang ada. Keterangan jarak terpendek dapat dilihat pada *listbox* dan pada gambar graf ditunjukkan dengan garis berwarna merah.

Selain mengetahui jarak terpendek, sistem ini dibuat untuk mencari estimasi waktu tempuh dari jarak terpendek yang didapat dengan cara klik tombol "Kecepatan" dan masukkan kecepatan kendaraan dalam satuan Km/jam, kemudian klik tombol "Waktu" untuk mengetahui estimasi waktu tempuh menuju tempat tujuan. Pada studi kasus diatas dimisalkan kecepatan 60 Km/jam dan diperoleh hasil estimasi waktu tempuh dari pantai Parangtritis menuju Candi Prambanan yaitu 46 menit. Lebih jelasnya dapat dilihat pada gambar 3 di bawah ini.



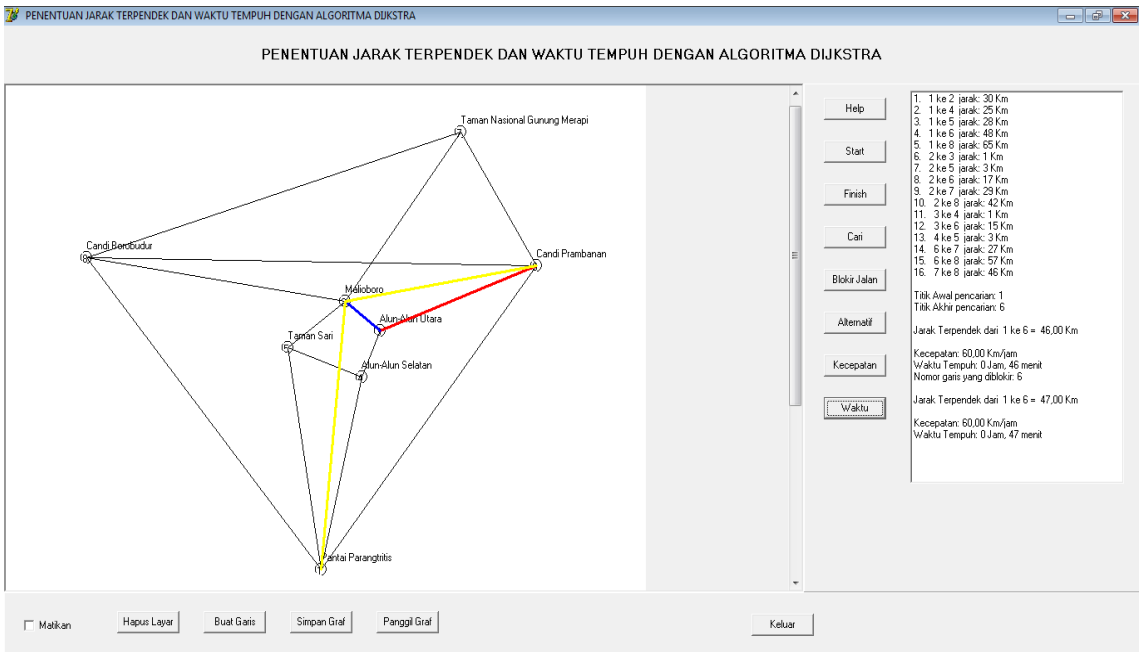
Gambar 3 : Tampilan jarak terpendek dan waktu tempuh

Setelah melakukan pencarian jarak terpendek, dilakukan pencarian jarak terpendek alternatif dengan uji coba pemblokiran salah satu sisi jalan pada jalur terpendek yang sudah ditemukan untuk mengetahui apakah sistem mampu menemukan jarak terpendek alternatif. *User* harus klik tombol "Blokir Jalan" dan memasukkan nomor garis yang akan diblokir. Keterangan nomor garis dapat dilihat pada *listbox* dan pada gambar graf ditunjukkan dengan garis berwarna biru. Tampilan pemblokiran jalan dapat dilihat pada gambar 4 di bawah ini.



Gambar 4 : Tampilan pemblokiran jalan

Pada saat terjadi pemblokiran jalan, sistem ini dapat menemukan jarak terpendek alternatif lain yang dapat ditempuh dengan cara klik tombol "Alternatif", sistem secara otomatis dapat menunjukkan jalur terpendek alternatif yang ditunjukkan garis berwarna kuning, jarak terpendek alternatif dapat dilihat pada gambar 5 di bawah ini. Setelah sistem menemukan jarak terpendek alternatif, estimasi waktu tempuh dari jarak terpendek alternatif dicari dengan cara yang sama seperti pencarian estimasi waktu tempuh sebelumnya.



Gambar 5: Tampilan jarak terpendek alternatif dan estimasi waktu tempuh

5. PENUTUP

Setelah dilakukan perancangan, implementasi dan uji coba sistem penentuan jarak terpendek dan waktu tempuh dengan algoritma dijkstra dapat diambil kesimpulan yaitu sistem yang dibuat dapat membantu menemukan jarak terpendek dan jarak terpendek alternatif apabila terjadi pemblokiran jalan pada jalur terpendek utama, selain itu sistem ini dapat juga digunakan untuk mengetahui estimasi waktu yang ditempuh. Kekurangan sistem ini terletak pada desain yang masih sederhana sehingga diperlukan pengembangan *interface* agar lebih menarik dan *user friendly*.

DAFTAR PUSTAKA

[1] R. A. D. Novandi, "Perbandingan Algoritma Dijkstra dan Algoritma Floyd-Warshall dalam Penentuan Lintasan (*Single Pair Shortest Path*)", Bandung : Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika Institut Teknologi Bandung. 2007.
[2] D. Joyner, M. V. Nguyen and N. Cohen, "Algorithmic Graph Theory Version 0.3", 2010, ch. 2, pp. 32.
[3] D. Joyner, M. V. Nguyen and N. Cohen, "Algorithmic Graph Theory Version 0.3", 2010, ch. 2, pp. 33-34.
[4] B. Stroustrup, "The C++ Programming Language 3nd Edition", Massachusett : Addition Weasley Longman Publishing Company, 1997, ch. 2, pp. 22.
[5] Borland® Delphi™ 7, "Developer's Guide", Scotts Valley : Borland Software Corporation, ch. 2, pp.2-1.
[6] D. Joyner, M. V. Nguyen and N. Cohen, "Algorithmic Graph Theory Version 0.3", 2010, ch. 2, pp. 29.
[7] I. P. Defindal, B. Ariesanda and Christoforus, "Algoritma G*reedy* untuk Menentukan Lintasan Terpendek", Bandung : Laboratarium Ilmu dan Rekayasa Komputasi, Departemen Teknik Informatika, Institut Teknologi Bandung.